

NIMBERS ARE INEVITABLE

JULIEN LEMOINE - SIMON VIENNOT

ABSTRACT. This article concerns the resolution of impartial combinatorial games, and in particular games that can be split in sums of independent positions. We prove that in order to compute the outcome of a sum of independent positions, it is always more efficient to compute separately the numbers of each independent position than to develop directly the game tree of the sum. The concept of number is therefore inevitable to solve impartial games, even when we only try to determinate the winning or losing outcome of a starting position. We also describe algorithms to use numbers efficiently and finally, we give a review of the results obtained on two impartial games: Sprouts and Cram.

1. INTRODUCTION

The games considered in this article are *combinatorial games*: two players play alternately, with perfect knowledge of the current state of the game, and there is no room for chance. We will restrain our discussion to *impartial* combinatorial games: from a given position, the same moves are available to both players.

Moreover, the theorems and algorithms of this article apply only to impartial combinatorial games in their *normal* version, where the first player who cannot move loses, and not to the *misère* version, where the first player who cannot move wins¹.

We will focus our attention particularly on *splittable* impartial games, in which some of the positions can be split in sum of independent positions. Our purpose is to *solve* these games, i.e. to find which player has a winning strategy and to compute it explicitly. In section 2, we review some background notions on impartial games, illustrating them with the games of Sprouts and Cram, and in section 3, we give some insight on the central concept of number.

In section 4, we develop the main result of this article: we prove that numbers are necessary when we try to compute the outcome of a splittable impartial game. In section 5, we detail algorithms to use numbers efficiently and finally, in section 6, we present the results obtained on the games of Sprouts and Cram.

2. BACKGROUND

2.1. Sprouts and Cram. The algorithms described in this paper can be applied to any impartial combinatorial game played in the normal version, and we have chosen two well-known games for our computations: Sprouts and Cram.

The game of Sprouts starts with a given number of spots drawn on a sheet of paper. Alternately, the players draw a line between two spots (possibly the same

¹The analysis of impartial games in misère version is much more complicated, notably because the algorithms described in this article cannot be applied.

spot), and add a spot anywhere on this line. The lines cannot cross each other, and a given spot cannot be used in more than 3 lines.

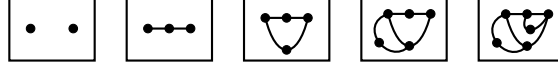


FIGURE 1. Example of a Sprouts game, starting with 2 spots (the second player wins).

The first article about Sprouts is from Martin Gardner [4] in 1967. A detailed presentation can be found in Winning Ways [2].

The game of Cram is played on a board with very simple rules: players alternately fill two adjacent cells with a domino, until one of them cannot play anymore. A description and some interesting analysis can also be found in [2].

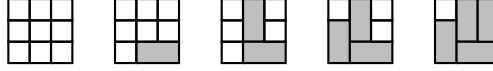


FIGURE 2. Example of a Cram game on a 3×3 board (the second player wins).

2.2. Splittable positions. Sprouts and Cram are *splittable* games, because some of the positions can be split in a sum of independent positions. When a player moves in such a position, the move can only affect one of the component of the sum, leaving the others untouched.

For example, the position of Sprouts on figure 3 is splittable. The spots at the interface between regions A and B cannot be used anymore, and any further move must be done inside the region A (without affecting B) or inside the region B (without affecting A).

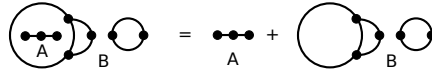


FIGURE 3. Splittable Sprouts position.

Figure 4 gives another example. The position is obtained after playing two moves in a Cram game on a 3×5 board. The position is splittable, because the two components are independent.

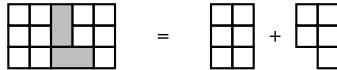


FIGURE 4. Splittable Cram position.

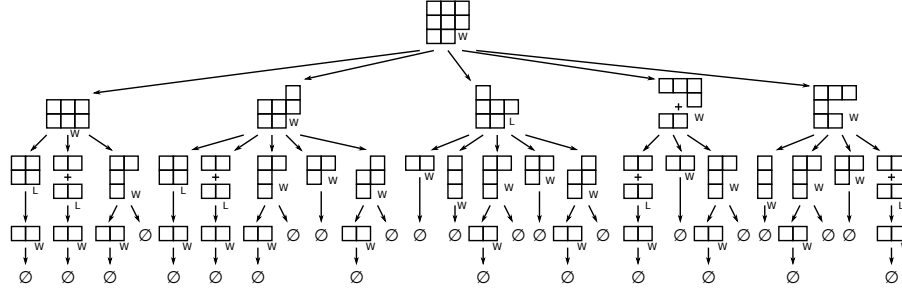


FIGURE 5. Game tree of a Cram position.

2.3. Game tree. We call *game tree of a position* \mathcal{P} the tree where nodes are the positions obtained by playing moves in \mathcal{P} , and in which two positions \mathcal{P}_1 and \mathcal{P}_2 are linked by an edge if \mathcal{P}_2 is an *option* of \mathcal{P}_1 (i.e. when \mathcal{P}_2 can be reached from \mathcal{P}_1 in one move).

The game tree of figure 5 has been obtained by identifying similar positions respectively to symmetry, and deleting isolated cells (since they cannot be used in any further move).

2.4. Outcome of a position. The *outcome* of a position is “W” (Win) if, from this position, the player to move has a winning strategy. Otherwise, the outcome is “L” (Loss). Positions whose outcome is “W” are said to be *winning* and those whose outcome is “L” are said to be *losing*.

It is possible to determine recursively the outcome of a position from its game tree. If a position \mathcal{P} has an option which is losing, then \mathcal{P} is winning. Otherwise, all options of \mathcal{P} are winning, and \mathcal{P} is losing. Finally, since the player who cannot move loses, the leaves (terminal positions) are losing.

The outcome of all the positions of figure 5 have been indicated.

2.5. Solution tree. The definition of the outcome of a position shows that it is sufficient to find only one losing option in order to prove that a node is winning. It implies that it is possible to determine the outcome of the root of the tree (the starting position) without knowing the outcome of all the positions of the tree.

On figure 6, we have selected a subset of the nodes from figure 5, which is sufficient to prove that the root is losing. There are 3 winning nodes for which it was not necessary to compute the outcome of all the options.

Such a subset of the complete game tree will be called a *solution tree* for the root. A solution tree is a graphic representation of what is also called a “winning strategy”. If the root is winning, like on figure 6, then the first player has a winning strategy. If the root is losing, the winning strategy is on the contrary for the second player.

The main goal of this article is to describe efficient methods to *solve* splittable impartial games, i.e. to compute a solution tree for the starting positions of these games.

2.6. Outcome of a sum of positions. When a position is split in a sum of independent components, it is sometimes possible to compute the outcome of components separately and deduce the result of the sum by using the following result (which can be proved easily by induction):

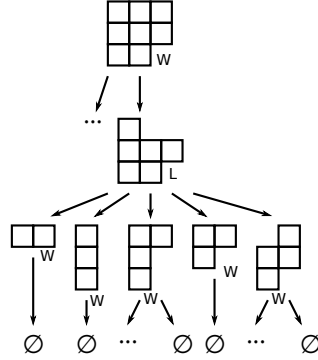


FIGURE 6. Solution tree for a Cram position.

Proposition 1. *The sum of two losing positions is losing. The sum of a losing position and a winning position is winning.*

This result allows us to reduce the size of the solution tree when a splittable position is computed. It was used for example by Applegate, Jacobson and Sleator in 1991 to compute the outcome of Sprouts positions [1]. However, it should be noted that it indicates nothing if both components are winning. To determine the outcome of the sum with separate computations even in this case, we need to use the concept of *nimber*.

3. NIMBER

3.1. The game of Nim. The game of Nim is played with heaps of objects, for example matches. A move consists in removing some of the matches from a single heap, and when the game is played in the normal version, the player that removes the last match wins (because the other player cannot play anymore).

A Nim-heap with n matches will be denoted \mathfrak{n} . The position $\mathfrak{7} + \mathfrak{5} + \mathfrak{4} + \mathfrak{2}$ is then composed of 4 heaps with 7, 5, 4 and 2 matches (respectively). For example, the player to move could choose to remove 3 matches in the second heap, which would lead to the position $\mathfrak{7} + \mathfrak{2} + \mathfrak{4} + \mathfrak{2}$. Or he could remove all the matches of the third heap, obtaining $\mathfrak{7} + \mathfrak{5} + \mathfrak{0} + \mathfrak{2}$.

Since a move is restricted to a single heap, the heaps are independent components and the game of Nim is a splittable game. A position from the game of Nim is then the sum of its heaps, which each are an independent component.

The resolution of Nim has been first described by Bouton, in 1902 [3]. The method uses the \oplus operator (*bitwise exclusive or*), which we will call *Nim-sum*. To compute the Nim-sum of two integers, we can write them in a binary form, and add the bits with the addition of $\mathbb{Z}/2\mathbb{Z}$ ($0 + 0 = 0$, $0 + 1 = 1$ and $1 + 1 = 0$). For example, $9 \oplus 12$ can be written in binary form $1001 \oplus 1100$, which gives 0101 (binary form). Back to the decimal usual notation, we obtain: $9 \oplus 12 = 5$.

The solution of Bouton can be stated as follows:

Theorem 1 (Bouton). *A sum of Nim-heaps has the same outcome as the Nim-sum of the heaps.*

Since the outcome of a single heap is L when it is empty, and W if there are still some matches left (just take all the matches), the losing positions of the complete game of Nim are those for which the Nim-sum of the heaps is 0.

Going back to our first example, it means that the position $7 + 5 + 4 + 2$ is winning, because $7 \oplus 5 \oplus 4 \oplus 2 = 4$. The winning moves are those that leave your opponent in a losing situation, and you should then remove matches so that you get $3 + 5 + 4 + 2$ (since $3 \oplus 5 \oplus 4 \oplus 2 = 0$), or $7 + 1 + 4 + 2$, or $7 + 5 + 0 + 2$.

3.2. Indistinguishability. We will say that two positions \mathcal{P}_1 and \mathcal{P}_2 are indistinguishable, and will denote $\mathcal{P}_1 \sim \mathcal{P}_2$ if, for any position \mathcal{P} , the sums of positions $\mathcal{P}_1 + \mathcal{P}$ and $\mathcal{P}_2 + \mathcal{P}$ have the same outcome. In this definition, the positions \mathcal{P}_1 , \mathcal{P}_2 and \mathcal{P} can be taken from any impartial combinatorial game.

This theoretical concept could be useful for practical computations. If a complicated position is known to be indistinguishable from a simpler one, we could replace the complicated one by the simple one in any sum appearing in the computation. For example, the figure 7 shows how we could accelerate the computation of figure 4, if we already knew that the position on the left is indistinguishable of a simpler position (with only two cells). This basic idea will be used in a more sophisticated way with the numbers.

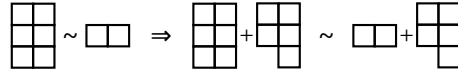


FIGURE 7. A Cram position simplified using indistinguishability.

3.3. Nimber. Here is the main result of the theory of impartial games:

Theorem 2 (Sprague-Grundy). *Any position of an impartial game played in the normal version is indistinguishable of some Nim-heap, called its nimber.*

A proof can be found, for example, in Winning Ways [2]. The following propositions can be deduced immediately:

Proposition 2. *Let \mathcal{P} a position.*

- * \mathcal{P} is losing \Leftrightarrow the nimber of \mathcal{P} is 0.
- * \mathcal{P} is winning \Leftrightarrow the nimber of \mathcal{P} is ≥ 1 .

Proposition 3. *Let \mathcal{P} a position.*

- * $\mathcal{P} \sim \mathfrak{n} \Leftrightarrow \mathcal{P} + \mathfrak{n}$ is losing.
- * $\mathcal{P} \sim \mathfrak{n} \Leftrightarrow \mathcal{P} + \mathfrak{n}$ is winning.

The nimber² has a practical interest in the case of a sum of independent positions. Indeed, it is possible to compute the nimber of the sum from the nimbers of the components, with the Nim-sum.

For example, on figure 4, the nimber of the first component is 1, and the nimber of the other is 0, so the nimber of the sum is 1 (since $1 \oplus 0 = 1$), and we can then deduce that the sum is winning. Note that this result could have been deduced with proposition 1. Now, let us consider figure 3. The nimber of each component

²The nimber is also called *number* or *function* of Sprague-Grundy.

is 2, so the number of the sum is 0 (since $2 \oplus 2 = 0$), and we can deduce that the sum is losing, which was not possible with only proposition 1.

Noting that $m \oplus n = 0 \Leftrightarrow m = n$, we obtain:

Proposition 4. *Let \mathcal{P}_1 and \mathcal{P}_2 two positions.*

- * $\mathcal{P}_1 + \mathcal{P}_2$ is losing \Leftrightarrow the numbers of \mathcal{P}_1 and \mathcal{P}_2 are equal.
- * $\mathcal{P}_1 + \mathcal{P}_2$ is winning \Leftrightarrow the numbers of \mathcal{P}_1 and \mathcal{P}_2 are different.

To complete this review of the concept of number, we need to explain how to compute the number of a position that is not a sum of independent components. We will use the following definition:

Definition 1. *We define the *Mex* (minimum excluded value) of a set of integers as the least positive integer that is not included in the set.*

For example, by applying this definition to numbers, we obtain: $Mex(1, 4) = 0$, $Mex(0, 1, 2, 5) = 3$.

The following proposition allows us to compute recursively the number of a position, knowing that the number of a terminal position is always 0:

Proposition 5. *The number of a position is equal to the *Mex* of the numbers of its options.*

We can now determine the numbers of all the positions of figure 5, as shown in the figure 8.

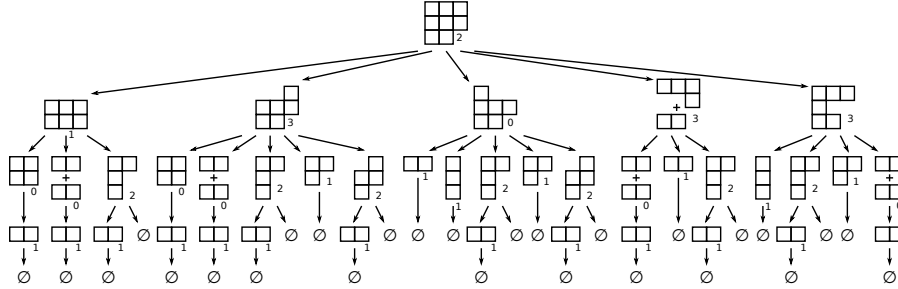


FIGURE 8. Nimbers of the game tree of a Cram position.

Since the definition of *Mex* uses all the options of a given position, it is sometimes assumed that it is needed to develop the complete game tree of a position in order to compute its number. For this reason, numbers are sometimes considered to require too much running time when we are only trying to compute the winning or losing outcome of a given starting position. The next section will show that, surprisingly, this is not the case at all.

4. NUMBERS ARE INEVITABLE

4.1. Elementary computation of the outcome of a sum of positions. The most simple way to compute the outcome of a sum of positions $\mathcal{P}_1 + \mathcal{P}_2$ is to consider the complete sum $\mathcal{P}_1 + \mathcal{P}_2$ as a single position, and compute the outcome of the options.

The figure 9 is an example of this method. Note that all the sums appearing in this solution tree – in particular the starting position – are of the kind $W+W$, and proposition 1 is then of no help to simplify the computation of the outcome.

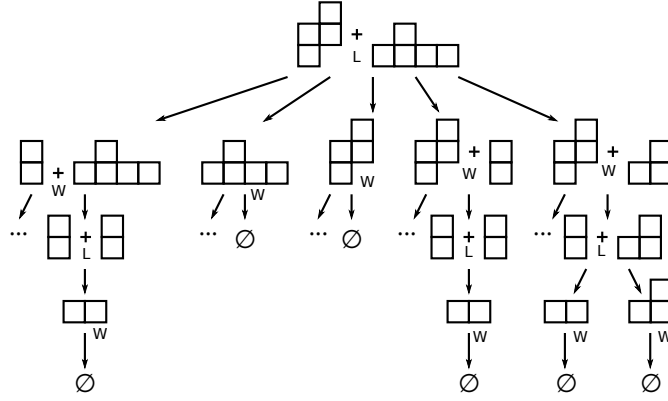


FIGURE 9. Solution tree for a sum of independent positions.

4.2. Inevitability of the nimbers. We can now state the main theoretical result of this article, which shows that even when we are only computing the outcome of a sum of positions, it is always more efficient to compute separately the nimbers of the components.

Theorem 3 (Inevitability of the nimbers). *Let us suppose that we have computed a solution tree for the sum $\mathcal{P}_1 + \mathcal{P}_2$ of two independent positions. Then, without computing any other node, we can determine the number of one component, and whether the number of the other component is equal or different.*

Proof. This result is proved by induction:

- * Terminal case: if $\mathcal{P}_1 = \emptyset$ and $\mathcal{P}_2 = \emptyset$, $\mathcal{P}_1 + \mathcal{P}_2$ is losing, and the number of both \mathcal{P}_1 and \mathcal{P}_2 is 0.
- * Induction:
 - Case 1: if $\mathcal{P}_1 + \mathcal{P}_2$ is winning, then one of the option is losing, for example of the form $\mathcal{P}_1^i + \mathcal{P}_2$. In that case, the number of \mathcal{P}_2 is known by induction hypothesis. And since $\mathcal{P}_1 + \mathcal{P}_2$ is winning, the numbers of \mathcal{P}_1 and \mathcal{P}_2 are different (with proposition 4).
 - Case 2: if $\mathcal{P}_1 + \mathcal{P}_2$ is losing, then all the options are winning. In particular, all the options of the form $\mathcal{P}_1^i + \mathcal{P}_2$ are winning. Either we know the number of \mathcal{P}_2 , or we know the numbers of all \mathcal{P}_1^i by induction hypothesis, from which we can deduce the number of \mathcal{P}_1 . And since $\mathcal{P}_1 + \mathcal{P}_2$ is losing, we conclude that the numbers of \mathcal{P}_1 and \mathcal{P}_2 are equal (with proposition 4 again).

□

This result shows that computing the nimbers of the components of the sum (more precisely computing the number of one component, and whether the other component has the same number or not) does not require to compute more positions than the elementary computation described in paragraph 4.1. And on the contrary, in most cases, it requires strictly less nodes. For example, instead of developing the tree of figure 9, it is more efficient to compute separately the nimbers of each independent component as in the figure 10. The number of both positions is 2, so

the number of the sum is \emptyset (since $2 \oplus 2 = 0$), from which we deduce finally that the sum is losing.

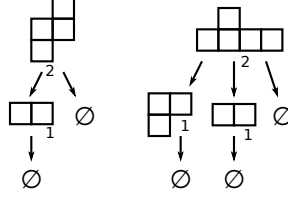


FIGURE 10. Computation of the numbers of the sum components.

Despite the simplicity of the demonstration, this result is surprising. Up to now, it was widely assumed that the inherent complexity of the concept of number made it unsuitable when trying to compute only the outcome of a sum of very complicated positions³. Yet, the theorem of inevitability proves that the elementary computation of the sum contains at least the computation of one number ! It is then always more efficient to use numbers as an intermediate step in order to compute the outcome of a sum of positions, and that, whatever complicated the positions of the sum are.

4.3. Computation of the number from a partial knowledge of the game tree. The theorem of inevitability shows that the computation of the number of one component is inevitable. However, it does not imply that all the ways of computing this number are equivalent: a direct use of the rule of the *Mex*, for example, would cause the development of the complete game tree, which is extremely costly in running time in case of a complicated position. In fact, it is possible, with the algorithms described in the next section, to compute the number of a position without developing the whole game tree. We give in figure 11 a sub-tree of figure 8, which is sufficient to determine the number of the root. Note in particular that for some nodes, we only prove that their number is different from a given value.

5. COMPUTATION ALGORITHMS

5.1. Reformulating number computations as outcome computations. In this section, we will detail algorithms that use the number to accelerate the computation of the outcome of sums of positions. We have to deal with two different kinds of computations: computations of outcomes, and computations of numbers. But the proposition 3 shows that it is equivalent to compute that $\mathcal{P} \sim n$ (i.e. to compute that the number of \mathcal{P} is n), or to compute that the outcome of $\mathcal{P} + n$ is L. Similarly, it is equivalent to compute that $\mathcal{P} \approx n$, or that the outcome of $\mathcal{P} + n$ is W. In this way, we can compute whether the number of a position is n or not, simply by computing the outcome of $\mathcal{P} + n$.

In the concrete implementation, we will represent $\mathcal{P} + n$ by a *couple* (\mathcal{P}, n) . We call \mathcal{P} the *position part* of the couple, and n the *number part*.

The options of a couple (\mathcal{P}, n) are of two kinds:

- * those of the position part, of the form (\mathcal{P}^i, n) where \mathcal{P}^i is an option of \mathcal{P} .
- * those of the number part, of the form (\mathcal{P}, i) with $i < n$.

³See for example the discussion at the end of the article of Applegate et al. [1].

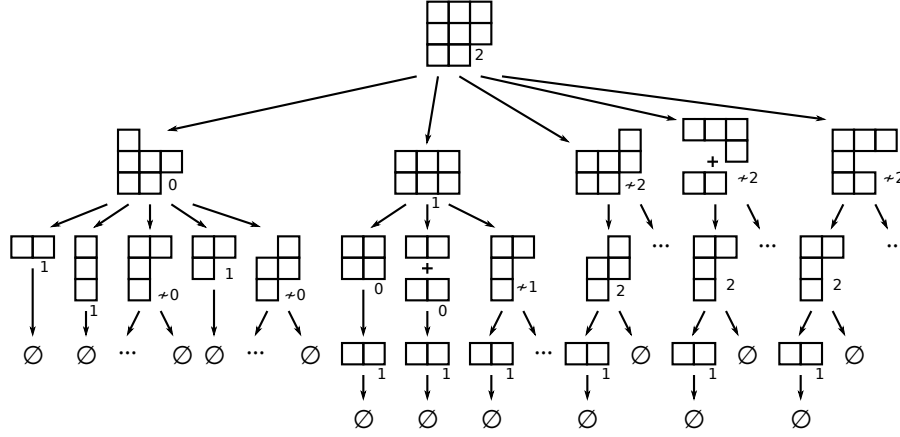


FIGURE 11. Part of the game tree sufficient to compute the number of the root.

The computation starts on the couple $(\mathcal{P}, 0)$. Indeed, this couple has no option in the number part, so we can identify the options of $(\mathcal{P}, 0)$ and \mathcal{P} . Note that this is true for any position \mathcal{P} and not only the starting position: we can identify $(\mathcal{P}, 0)$ and \mathcal{P} , and in particular they have the same outcome.

5.2. Computation of the outcome of a couple. The key-point in the case of a splittable game is to check whether \mathcal{P} is splittable or not before computing recursively the outcome of (\mathcal{P}, n) . If the position is splittable, we use algorithm 2 described thereafter.

Algorithm 1 (Recursive computation of the outcome of a couple).

To compute the outcome of the couple (\mathcal{P}, n) :

- * If \mathcal{P} is splittable, compute the outcome of the couple with algorithm 2, otherwise:
- * For each option (\mathcal{P}^i, n) of the position part and each option (\mathcal{P}, i) of the number part, compute the outcome of the option with algorithm 1.
If the option is losing, return “W”.
- * If all the options are winning, return “L”.

Let us note that in the case of a non-splittable position \mathcal{P} , and by using $n = 0$ as the number part, the algorithm is the same as the classical algorithm used to compute the outcome of \mathcal{P} .

5.3. Computation of the outcome of a sum. To compute the outcome of a couple when the position part is splittable in a sum of the form $(\mathcal{P}_1 + \dots + \mathcal{P}_k, n)$, we first compute the numbers of all the components except one, with algorithm 3 described thereafter. Then, we merge the numbers with the Nim-sum. The couple is therefore reduced to a couple of the form (\mathcal{P}_k, n') , without any sum in the position part, and we compute its outcome with algorithm 1.

Algorithm 2 (Computation of the outcome of a sum).

To compute the outcome of a couple $(\mathcal{P}_1 + \dots + \mathcal{P}_k, n)$:

- * For j from 1 to $k - 1$, compute n_j , the number of \mathcal{P}_j , with algorithm 3.

- * Compute $\mathfrak{n}' = \mathfrak{n}_1 + \dots + \mathfrak{n}_{k-1} + \mathfrak{n}$ with the Nim-sum.
- * Compute the outcome of $(\mathcal{P}_k, \mathfrak{n}')$ with algorithm 1, and return the obtained value.

5.4. Computation of the number of the position. Lastly, we need to explain how to compute the number of a position, which was necessary in the previous algorithm. The principle is simply to try the numbers in increasing order: 0, then 1, then 2,... until we find the correct value.

Algorithm 3 (Computation of the number of a position).

To compute the number of a \mathcal{P} :

- * Initialise \mathfrak{n} to 0.
- * While the computation of the outcome of $(\mathcal{P}, \mathfrak{n})$ with algorithm 1 returns “W”, increment \mathfrak{n} .
- * Return the final value of \mathfrak{n} .

The returned value is the number of \mathcal{P} , since the loop ends when $(\mathcal{P}, \mathfrak{n})$ is found losing.

It should be noted that this algorithm has the advantage of avoiding useless computations, because the computation of $\mathcal{P} \sim \mathfrak{n}$ contains the computation of $\mathcal{P} \sim \mathfrak{k}$ for $\mathfrak{k} < \mathfrak{n}$. This comes from the fact that if we have proved that $\mathcal{P} \sim \mathfrak{n}$, i.e. we have proved that the couple $(\mathcal{P}, \mathfrak{n})$ is losing, then we have proved that each option of this couple is winning. In particular, for any $\mathfrak{k} < \mathfrak{n}$, the option $(\mathcal{P}, \mathfrak{k})$ is winning, which means that $\mathcal{P} \sim \mathfrak{k}$.

5.5. Game tree traversal. The efficiency of the algorithms described above depends on the path that we choose in the game tree. In the case of the classical algorithm for computing the outcome of a position, it is sufficient to find a losing option in order to prove that a position is winning. Therefore, the choice of the option that we compute first is important: if we choose a winning option before a losing one, there will be useless computations. And if there is more than one losing option, it is better to choose the “easiest” one first, in order to obtain a smaller solution tree, and to obtain it faster.

Such a choice is also needed in the algorithms of the previous section. Firstly, in algorithm 1: just as in the classical computation, if the couple $(\mathcal{P}, \mathfrak{n})$ is winning, it is better to search the game tree from the easiest losing option first. But a choice also occurs in algorithm 2: if the couple $(\mathcal{P}_1 + \dots + \mathcal{P}_k, \mathfrak{n})$ is winning, the number of one component will not be computed (in the description of the algorithm, it is \mathcal{P}_k , but we can choose any of the components). The choice of this component will affect the speed of the computation.

Of course, these choices are not easy to perform, because we do not know which option is the losing one - if we knew it there would be no need for computation ! It should be noted that even if the number plays a central role in the algorithms, using the notion of couple enables us to keep some kind of similarity with the classical algorithm to compute the outcome. It follows that most of the usual methods (Depth-First, or Best-First, like the PN-search) used to search game trees in an efficient order can be used in combination with the algorithms of this paper, with some adaptations.

6. RESULTS

6.1. Game of Sprouts. We have applied the algorithms described in the previous section to the game of Sprouts, which allowed us to compute the outcome of the game up to 32 starting spots and some sparse values up to 47 spots. The following table indicates, for a given number of starting spots p , the number of losing couples⁴ stored in the solution tree obtained at the end of the computation (after pruning useless positions). All the computed outcomes support the “Sprouts conjecture”: the position with p starting spots is losing if and only if $p = 0, 1$ or 2 modulo 6.

It is interesting to note that before the introduction of the algorithms described in this article, the biggest known outcome was $p = 11$, with more than 100,000 losing positions at the end of the computation [1], whereas we are now able to compute the same position with only 140 losing couples. The algorithms of this article are particularly efficient in the case of Sprouts because splittable positions are extremely frequent, and appear even in the upper part of the game tree.

p	size	p	size	p	size	p	size	p	size	p	size	p	size
0	0	7	103	14	1580	21	9270	28	14813	35	18812		
1	1	8	205	15	3252	22	5706	29	3414	...	?		
2	3	9	63	16	1068	23	2837	30	58363	40	45782		
3	6	10	140	17	471	24	9316	31	58365	41	48890		
4	16	11	140	18	3233	25	9229	32	58204	...	?		
5	38	12	475	19	3630	26	18567	33	?	47	54542		
6	64	13	577	20	4051	27	59117	34	21107	...	?		

FIGURE 12. Results obtained on the game of Sprouts.

The details about the optimisations specific to the game of Sprouts can be found in our article from 2007 [5] and the update published in 2010 [6].

6.2. Game of Cram. We have also applied the same algorithms to the game of Cram, and we present here the results obtained up to this point.

There exists a symmetry strategy on boards of even \times even dimensions, which are losing (and hence their nimber is 0), and similarly on boards of even \times odd dimensions, which are winning. But this strategy does not apply to odd \times odd boards, and tells nothing about the nimber of even \times odd boards (except that this nimber is not 0). In the tables below, we have indicated in parenthesis the results that does not need any computation because of the symmetry strategy. Moreover, we have indicated with “—” the $n \times m$ board where $n > m$, because the value is the same as on $m \times n$ boards, with a simple symmetry argument.

As far as we know, the best results known so far were those of Martin Schneider in 2009 [7], which we have indicated in the tables with a “*”.

3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
*0	*1	*1	*4	*1	*3	*1	2	0	1	2	3	1	4	0	1

FIGURE 13. Results obtained on $3 \times n$ boards.

⁴We store only losing couples, i.e. positions whose nimber is known, in order to save memory.

	4	5	6	7	8	9
4	(0)	*2	(0)	3	(0)	1
5	—	*0	2	*1	1	W
6	—	—	(0)	>3	(0)	(W)
7	—	—	—	W	(W)	

FIGURE 14. Results obtained on $n \times m$ boards, with $n \geq 4$ and $m \geq 4$.

The new results on boards with both dimensions greater than 4 are the numbers of the 4×7 , 4×9 , 5×6 , and 5×8 boards, and the winning outcome of the 5×9 and 7×7 boards. With algorithm 3, we have also been able to compute that the number of the 6×7 board is strictly greater than 3, but without achieving to compute the exact value up to now.

In the case of the game of Cram, we have noted experimentally that a slight increase in the board dimensions creates a huge increase of the computation difficulty. This is due of course to the exponential increase of the number of possible board positions, but also to the fact that the greater the board dimensions, the later the splitting of the positions occur in the game tree. The optimisations specific to the game of Cram will be the subject of a future article.

CONCLUSION

Since the discovery of the Sprague-Grundy theorem, the numbers have been used successfully to analyse a number of impartial games, in particular the numerous variants of Nim, like the octal games. However, in the case of very intricate games, like Sprouts or Cram, the numbers were usually considered to consume too much running time, and were not or almost not used to compute the outcome of the starting positions.

The theorem presented in this article shows on the contrary that the use of numbers in impartial splittable games is inevitable, even when we are only trying to compute the outcome of a starting position, because the elementary computation of the outcome of a sum of positions indeed computes the number of one of the component. Algorithms using numbers efficiently have been applied successfully to Sprouts and Cram, two impartial splittable games, and numbers play a central part in the results obtained.

REFERENCES

- [1] D. Applegate, G. Jacobson, and D. Sleator, *Computer Analysis of Sprouts*, Tech. Report CMU-CS-91-144, Carnegie Mellon University Computer Science Technical Report, 1991.
- [2] Elwyn Berlekamp, John Conway, and Richard Guy, *Winning ways for your mathematical plays*, A K Peters, 2001.
- [3] C. L. Bouton, *Nim, a game with a complete mathematical theory*, Ann. of Math. **3** (1902), 35–39.
- [4] Martin Gardner, *Mathematical games : of sprouts and brussels sprouts, games with a topological flavor*, Scientific American **217** (July 1967), 112–115.
- [5] Julien Lemoine and Simon Viennot, *A further computer analysis of sprouts*, <http://download.tuxfamily.org/sprouts/sprouts-lemoine-viennot-070407.pdf>, 2007.
- [6] ———, *Computer analysis of sprouts with numbers*, <http://arxiv.org/abs/1008.2320>, 2010.
- [7] Martin Schneider, *Das spiel juvavum*, <http://www.mschnneider.cc/papers/masterthesis.pdf>, 2009.